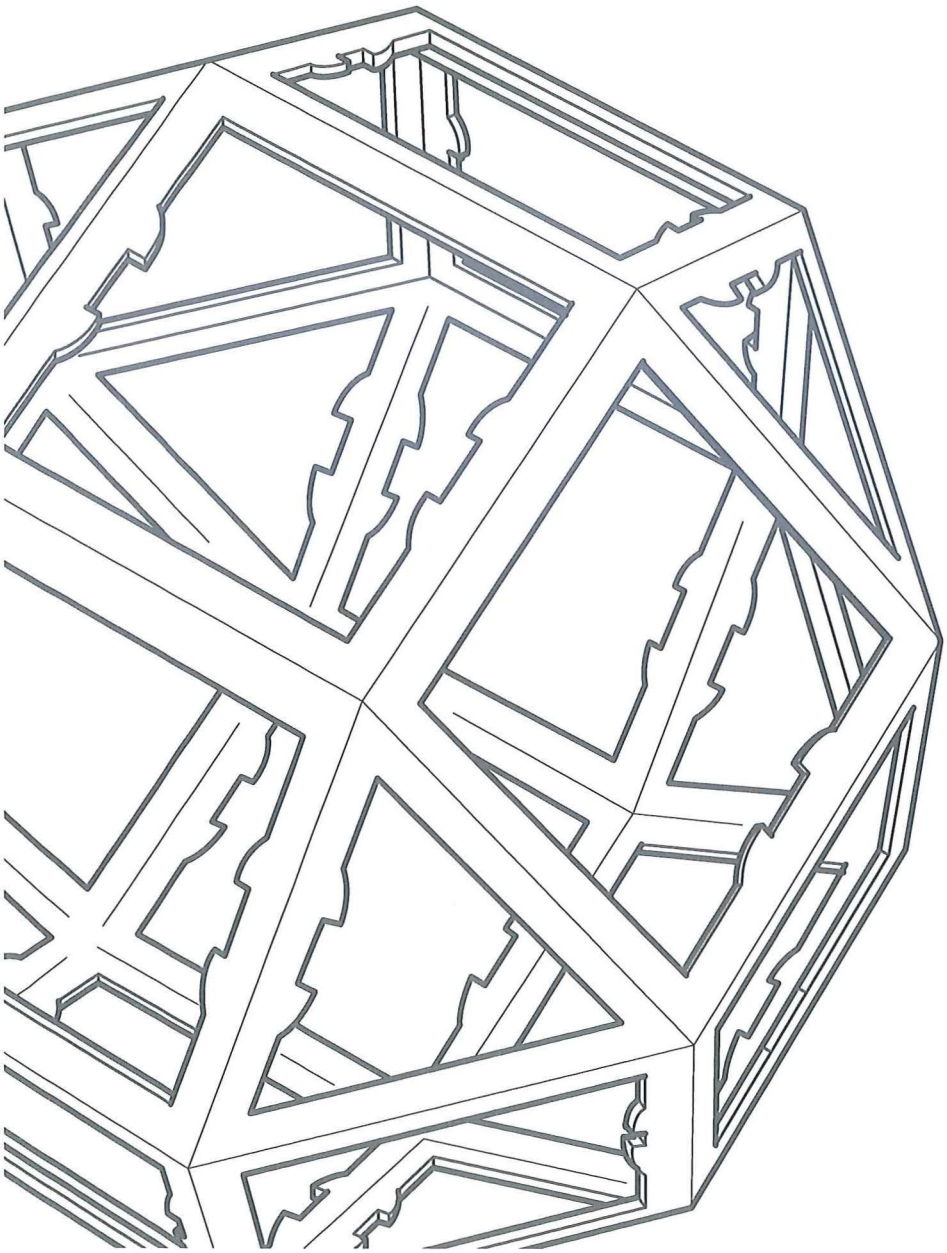
Department of Mathematics and Statistics
College of Engineering

UC
UNIVERSITY OF CANTERBURY
Te Whare Wānanga o Waitaha
CHRISTCHURCH NEW ZEALAND

Summer Research Project

Losing Lotto, A Mathematical Approach

by Michael Snook



08

2007/2008 : Summer Research Project
Losing Lotto, A Mathematical Approach

Michael Snook
Supervisor : Dr. Günter Steinke
Department of Mathematics,
University of Canterbury

February 8, 2008

Abstract

An (n, k, p, t) lotto design is a set of k sets (called blocks) of an n set such that any p set intersects at least one block in at least t points. We will denote the minimal number of blocks needed to make an (n, k, p, t) lotto design by $L(n, k, p, t)$. This paper lists a few known theorems for upper and lower bounds for lotto designs. We then apply these theorems to the New Zealand lotto system and calculate upper and lower bounds for each of the six divisions of the New Zealand system.

Contents

CHAPTER 1

Introduction

Lotteries are a common form of gambling. Generally they work by bettors choosing tickets with k numbers on them between 1 and n. The house then draws p random numbers between 1 and n. If a bettor has t or more numbers that are drawn on their ticket, then they win a prize. Usually for the lower values of t, the bettor wins a fixed amount of money. While for the larger values of t the bettor wins a share of the prize, depending on the number of other people who had the same number of numbers drawn on their tickets. The prize for having all k of your numbers being drawn is almost always a large sum of money. Because of this, everybody wants to win. There are numerous systems out there all claiming to increase your chances at winning. Some of these systems are based on *lotto designs*. The focus of this paper will be on lotto designs, and how they can/can not help us win the lottery. Chapter 2 will cover the basic definitions and ideas of lotto designs. This will include a number of theorems and their proofs. In the second part of chapter 2 will look at a few algorithms used to construct lotto designs.
The most common lottery in New Zealand is Lotto, run by the New Zealand Lotteries Commission. In chapter 3 we will be trying to apply the theorems and constructions in chapter 2 to the NZ lotto system. Our main focus will be trying to construct upper and lower bounds for the number of tickets needed to guarantee winning each of the six divisions of Lotto.

CHAPTER 2

Lotto, and other Designs

2.1 Lotto Designs

To begin with we will need a few definitions. We will begin with some basic combinatorics definitions, then move onto definitions used in making lotto designs.
Note: throughout this paper we will be referring to p sets, k sets, etc. These are just sets with p or k points in them respectively.

Definition 2.1. A *combinatorial design* is a pair $(\mathcal{P}, \mathcal{B})$, where $\mathcal{B}$ is a non-empty set of *blocks* made up of elements called *points* from a set $\mathcal{P}$.

Definition 2.2. A $t-(n,k,\lambda)$ *design* is a combinatorial design such that $\mathcal{P}$ has n points in it, every block in $\mathcal{B}$ has exactly k points, and such that every t element subset of $\mathcal{P}$ is contained in exactly λ blocks.

The invitation problem is a common combinatorial problem solved using combinatorial designs. The idea is to invite n friends to dinner, k at a time in such a fashion that everybody has dinner with everybody else at least once.

Example 2.3. The invitation problem for inviting 7 friends to dinner, 3 at a time can be represented by a $2-(7,3,1)$ design. In the case of this example $\mathcal{P} = \{1,2,3,4,5,6,7\}$, and the blocks of $\mathcal{B}$ are:

$$\{1,2,3\},\ \{1,4,5\},\ \{1,6,7\},\ \{2,4,6\},\ \{2,5,7\},\ \{3,4,7\},\ \{3,5,6\}.$$

Definition 2.4. An (n,k,p,t) lotto design is a set of blocks each consisting of k points from an n set such that every p set intersects a block in at least t points. We will denote the minimal number of blocks an (n,k,p,t) lotto design can have by $L(n,k,p,t)$.

Example 2.3 is an example of an $(7,3,2,2)$ lotto design. A special case of lotto designs is when you have a (n,k,k,k) lotto design. In this case the lotto design has to include every k subset of the set n. This makes $L(n,k,k,k)$ very easy to calculate. It is just $\binom{n}{k}$.

Example 2.5. A $(12,4,6,3)$ lotto design (on the set $N = \{1,..,12\}$) is given by the following 8 blocks:
$\{1,2,3,4\}, \{1,5,6,7\}, \{2,3,4,5\}, \{2,3,4,6\}, \{8,9,10,11\}, \{8,9,10,12\}$, $\{8,9,11,12\}, \{8,10,11,12\}$.

If we take any random 6 set A, then there is a block B such that $|A \cap B| \geq 3$. For example, if $A = \{1, 4, 5, 7, 8, 10\}$, then $B = \{1, 5, 6, 7\}$. If we take the intersection of these two sets, then we get $\{1, 4, 5, 7, 8, 10\} \cap \{1, 5, 6, 7\} = \{1, 5, 7\}$.

It is very important to note that while it seems lotto designs give you an advantage this is not true. They do not improve the expected return on each ticket. However, they do increase your odds of winning something small instead of winning nothing at all.

There are two important subclasses of lotto designs. These are covering designs, and Turán designs. A (n, k, t) covering design is a design such that choosing blocks of k points from a set of n points, any subset of t points is contained in one of those blocks. In terms of lotto designs, a covering design is an (n, k, t, t) lotto design. Any $t-(n, k, 1)$ design is a covering design. We will denote the minimal number of blocks in a (n, k, t) covering design by $C(n, k, t)$

A (n, p, t) Turán design is a design with blocks of k points chosen from a set of n points where any arbitrary subset of p points will contain at least one block of the design. A Turán design is an (n, t, p, t) lotto design. The minimal number of blocks in a Turán design will be denoted by $T(n, p, t)$. Both covering and Turán designs have been studied to a great extent as stand alone designs, and many papers have been written about them, see [4].

Covering and Turán designs are closely related as theorem 2.6 will show us.

Theorem 2.6. *The complement of a Turán design is a covering design, and the complement of a covering design is a Turán design. Consequently* $T(n, p, t) = C(n, n - t, n - p)$.

Proof. We will denote the set $\{1, .., n\}$ by N. Remember that a Turán design is a $L(n, t, p, t)$ design, so any p-set will contain a k set of $t = k$ elements of the Turán design. This can be seen in fig 2.1 where the k set B is shown in green, the p set A is shown in purple, and everything else is grey.

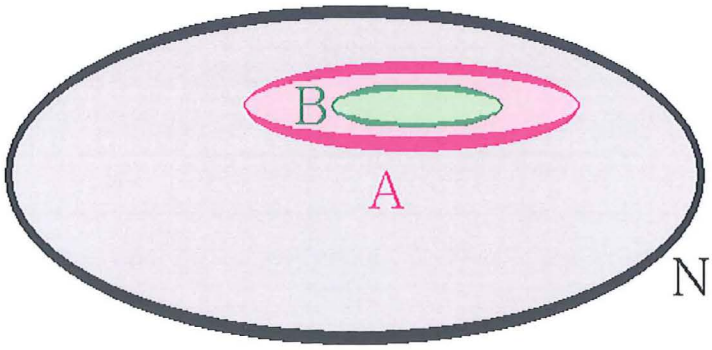


Figure 2.1: A block of a Turán design

If we take the complement of these sets we will get something that looks like fig 2.2, where $B' = N \backslash A$, and $A' = N \backslash B$. In this figure the $n - p$ set B' is

shown in green, the $n-t$ set A' is shown in purple, and everything else is grey.

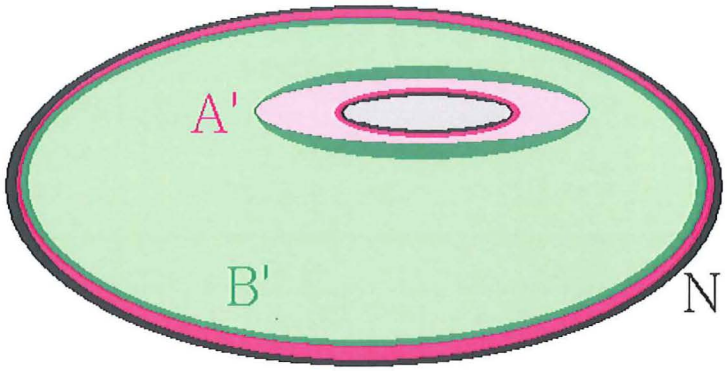


Figure 2.2: The complement of a block of a Turán design.

Take any $n-p$ set B' in the N, then there is a corresponding p set $N\backslash B' = A$. In the original Turán design there is a block B contained in A. Since $A \supseteq B$, we can conclude

$$B' = N\backslash A \subseteq N\backslash B = A'.$$

This works for any $n-p$ set B' you could choose, so we have for any $n-p$ set B' there is a $n-t$ block A' that contains it. Therefore we have a $(n, n-t, n-p)$ covering design. This shows us that the complement of a Turán design is a covering design and thus $T(n,p,t) \leq C(n, n-t, n-p)$. Using a similar argument, working backwards it can be shown that $T(n,p,t) \geq C(n, n-t, n-p)$. Therefore $T(n,p,t) = C(n, n-t, n-p)$.

□

It is often computationally difficult to construct large lotto designs. As the size of n increases, the size of the corresponding lotto design increases rapidly. Often it is better to construct upper and lower bounds for our desired lotto designs first. This approach allows us to find out whether our lotto design will be practical without calculating the actual lotto design. Once we have done this, if the design could be practical, we can then go about constructing the desired design. Doing this can save us a lot of time. Presented below are a number of theorems used to calculate upper and lower bounds for lotto designs. This is by no means a complete list, there are many more theorems out there that can be used to calculate lower and upper bounds for lotto designs, see [3] pages 578-584, [7], [8] and [9].

Theorem 2.7. *If* $n = n_1 + n_2$, *and* $p = p_1 + p_2 - 1$, *then* $L(n,k,p,t) \leq L(n_1,k,p_1,t) + L(n_2,k,p_2,t)$.

Proof. To show that this is true we will show that if you take any (n_1,k,p_1,t) and (n_2,k,p_2,t) lotto designs on disjoint sets, then when we combine them we have an (n,k,p,t) lotto design. This situation can be shown by fig 2.3,

where the sets p_1 and p_2 are in purple and the blocks B_1 and B_2 are in blue. In the figure, either $t_1 \geq t$ or $t_2 \geq t$ (or both).

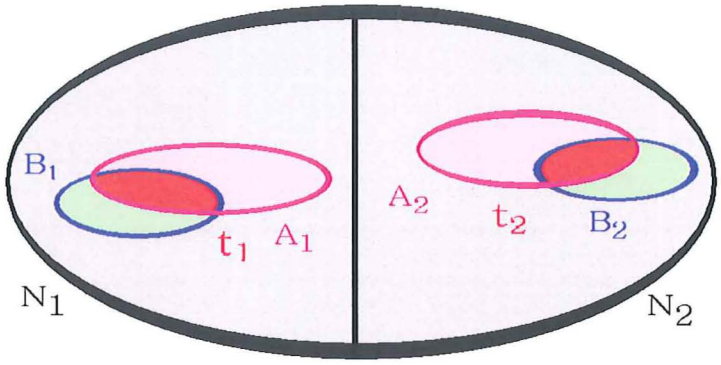


Figure 2.3:

Let $N = \{1, .., n\}$, $N_1 = \{1, .., n_1\}$ and $N_2 = \{n_1 + 1, .., n\}$. Also let $\mathcal{B} = \mathcal{B}_1 \cup \mathcal{B}_2$, where $\mathcal{B}_1$ is the set of blocks of an (n_1, k, p_1, t) lotto design on N_1 and $\mathcal{B}_2$ is the sets of blocks of an (n_2, k, p_2, t) lotto design on N_2. We will show that $(N, \mathcal{B})$ is an (n, k, p, t) lotto design on N. Take any p set $A \subseteq N$. Either $|A \cap N_1| \geq p_1$, or $|A \cap N_2| \geq p_2$. Otherwise we would have

$$|A| \leq |(A \cap N_1) \cup (A \cap N_2)| \leq p_1 - 1 + p_2 - 1 = p - 1.$$

This is a contradiction because $|A| = p$. If $|A \cap N_1| \geq p_1$, then there is a block $B_1 \subseteq \mathcal{B}_1$ such that $|A \cap B_1| \geq t_1$. If $|A \cap B_2| \geq p_2$, then there is a block $B_2 \subseteq \mathcal{B}_2$ such that $|A \cap N_2| \geq t_2$. Either way there is a block $B \subseteq \mathcal{B}$ such that $|B \cap A| \geq t$. Therefore we have an (n, k, p, t) lotto design.

□

Example 2.8. Say we are wanting to find an upper bound for $L(27, 6, 8, 3)$. Using theorem 2.3 we know

$$L(27, 6, 8, 3) \leq L(15, 6, 4, 3) + L(12, 6, 5, 3).$$

From the tables in [6] we know that $L(12, 6, 5, 3) = 2$, and $L(15, 6, 4, 3) \leq 14$. This gives us

$$L(27, 6, 8, 3) \leq L(15, 6, 4, 3) + L(12, 6, 5, 3) \leq 14 + 2 = 16.$$

Theorem 2.9. $T(n, p, t) \geq \frac{\binom{n}{t}}{\binom{p-1}{t-1}} \cdot \frac{n-p+1}{n-t+1}$.

Proof. This is just a rewording of the proof from [2]. To start off we show that:

$$\frac{\binom{n}{t}}{\binom{p-1}{t-1}} \cdot \frac{n-p+1}{n-t+1} = \frac{n-p+1}{t\binom{p-1}{t-1}} \cdot \binom{n}{t-1}.$$

This is relatively straightforward, and goes as follows.

$$\begin{aligned}
\frac{\binom{n}{t}}{\binom{p-1}{t-1}} \cdot \frac{n-p+1}{n-t+1} &= \frac{n!/((n-t)!\cdot t!)}{(p-1)!/((p-t)!\cdot(t-1)!)} \cdot \frac{n-p+1}{n-t+1} \\
&= \frac{n!\cdot(p-t)!\cdot(t-1)!}{(n-t)!\cdot t!\cdot(p-1)!} \cdot \frac{n-p+1}{n-t+1} \\
&= \frac{n!}{(n-t+1)!\cdot(t-1)!\cdot t} \cdot \frac{(p-t)!\cdot(t-1)!}{(p-1)!} \cdot (n-p+1) \\
&= \frac{\binom{n}{t-1}}{t} \cdot \frac{1}{\binom{p-1}{t-1}} \cdot (n-p+1) \\
&= \frac{n-p+1}{t\cdot\binom{p-1}{t-1}} \cdot \binom{n}{t-1}
\end{aligned}$$

Let H be a collection of t subsets of the set $N = \{1, .., n\}$. We will define m_p to be the number of distinct collections of all possible t subsets of p subsets of N such that every t set in the collection is contained in H. That is, if G is such a collection, then there exists a p set such that every possible t subset of the p set is a t set in H. The collection G would be all the t subsets of the p set. Note that m_t is just $|H|$, and that m_{t-1} is just $\binom{n}{t-1}$. Now, theorem 1 from [2] states that

$$m_{p+1} \geq \frac{p^2 \cdot m_p}{(p-t+1)(p+1)} \left(\frac{m_p}{m_{p-1}} - \frac{(t-1)(n-p)+p}{p^2} \right), \tag{2.1}$$

whenever $m_{p-1} \neq 0$. Let

$$F(n,p,t) = \frac{\binom{p-1}{t-1}(n-t+1) - (n-p+1)}{t\binom{p-1}{t-1}} \binom{n}{t-1}.$$

It can be shown through long and tedious, but simple calculations that

$$F(n,p+1,t) = F(n,p,t) + \frac{(t-1)(n-p)+p}{t^2\binom{p}{t}} \binom{n}{t-1} \tag{2.2}$$

We will now prove by induction on p that

$$m_p \geq m_{p-1} \frac{t^2\binom{p}{t}}{p^2\binom{n}{t-1}} (m_t - F(n,p,t)) \tag{2.3}$$

when $p = t$ the theorem does not apply because $m_{t-2} = 0$. When $p = t+1$, from (2.1) we have

$$\begin{aligned}
m_{t+1} &\geq \frac{t^2 m_t}{(t+1)m_{t-1}} \left(m_t - \frac{(t-1)(n-p)+p}{t^2} \binom{n}{t-1} \right) \\
&= m_t \frac{t^2\binom{t+1}{t}}{(t+1)^2\binom{n}{t-1}} (m_t - F(n,t+1,t)).
\end{aligned}$$

This proves the case $p = t+1$. From this we also see that $m_{t+1} > 0$ whenever $m_t > \frac{(t-1)(n-t)+t}{t^2}\binom{n}{t-1}$.
Now, if $p > t+1$ then we have

$$\begin{aligned}
& & m_{p+1} & \\
& by(2.1) & \geq & m_p \frac{p^2}{(p-t+1)(p+1)} \left(\frac{m_p}{m_{p-1}} - \frac{(t-1)(n-p)+p}{p^2} \right) \\
& induction(2.3) & \geq & m_p \frac{p^2}{(p-t+1)(p+1)} \left(\frac{t^2}{p^2} \frac{\binom{p}{t}}{\binom{n}{t-1}} (m_t - F(n,p,t)) - \frac{(t-1)(n-p)+p}{p^2} \right) \\
& & = & m_p \frac{t^2\binom{p}{t}}{(p+1)(p-t+1)\binom{n}{t-1}} \left(m_t - \left[F(n,p,t) + \frac{(t-1)(n-p)+p}{t^2\binom{p}{t}} \binom{n}{t-1} \right] \right) \\
& by(2.2) & = & m_p \frac{t^2\binom{p+1}{t}}{(p+1)^2\binom{n}{t-1}} (m_t - F(n,p+1,t))
\end{aligned}$$

This proves (2.3). It follows from (2.3) that $m_p > 0$ whenever $m_t > F(n,p,t)$. In other words, H contains all t subsets of some p subset of N if $|H| > F(n,p,t)$. Let $N^{(t)}$ be the set of all possible t subsets of the set N. The collection of sets $\overline{H} = N^{(t)} \backslash H$ will have a p subset which does not contain any t subset from $\overline{H}$ if $|\overline{H}|$ is strictly less then $\binom{n}{t} - F(n,p,t)$. Therefore

$$T(n,p,t) \geq \binom{n}{t} - F(n,p,t) = \frac{(n-p+1)}{t\binom{p-1}{t-1}} \binom{n}{t-1} = \frac{\binom{n}{t}}{\binom{p-1}{t-1}} \cdot \frac{n-p+1}{n-t+1}.$$

□

The next theorem is useful for creating lower bounds for lotto designs. The lower bounds in chapter 3 where calculated using theorem 2.9 with the following theorem.

Theorem 2.10. $L(n,k,p,t) \geq \frac{T(n,p,t)}{\binom{k}{t}}$

Proof. This can be rewritten as $L(n,k,p,t)\binom{k}{t} \geq T(n,p,t)$. Take a (n,k,p,t) lotto design with $L(n,k,p,t)$ blocks. Any p set will have at least t points in a k block. If we take every k block and replace it with $\binom{k}{t}$ blocks corresponding to all different combinations of t points made up from the k points in that block, then we will have at most $L(n,k,p,t)\binom{k}{t}$ blocks. Now take any p set, then there is a block in the original design that covered at least t points from this set. This set of t points is a possible combination of t points made from the k points of the block. Therefore there is a block of t points in the new design that contains these t points. That is, those t points form a block of the new design. This gives us a (n,t,p,t) lotto design, that is a (n,p,t) Turán design. Therefore $L(n,k,p,t) \geq \frac{T(n,p,t)}{\binom{k}{t}}$. □

Theorem 2.11. *Let* $p = \tau(t-1) + 1 + \upsilon$ *where* $0 \leq \upsilon \leq t-1$, *then*

$$L(n,k,p,t) \leq min_{n-\upsilon=\sum_{i=1}^{\tau}\sigma_i} \sum_{i=1}^{\tau} C(\sigma_i,k,t).$$

Before we provide a proof of this it would be a good idea to explain a little more about what this theorem is saying. In its current form it is a bit confusing. It simply states that given τ suitable coverings on disjoint sets, if we form the union of all the blocks of each covering design we get an (n,k,p,t) lotto design. The minimum is taken over all possible combinations of σ_i. We try all possible combinations of σ_i and find the one that gives the minimum sum. Figure 2.4 may make things clearer, we can see the set N partitioned into τ disjoint sets each with a covering on them plus the set of υ points which is disjoint from the other sets.

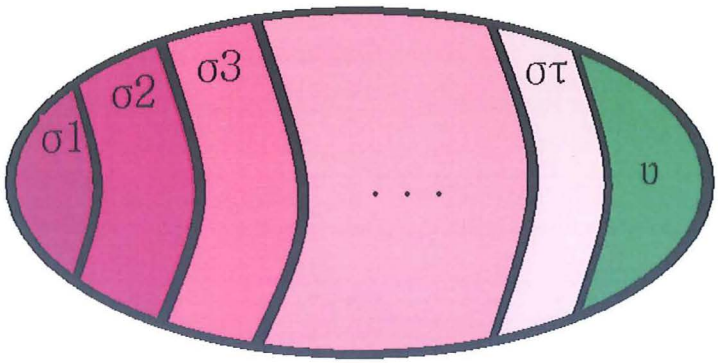


Figure 2.4: $(n,k,p,t) = \cup_{i=1}^{\tau}(\sigma_i,k,t)$

Now that we understand what is going on, we can provide a proof.

Proof. To prove this we need to show that if we have the union of (σ_i,k,t) coverings on disjoint sets such that $\sum_{i=1}^{\tau}\sigma_i + \upsilon = n$, then we also have a (n,k,p,t) lotto design. Suppose we have τ suitable coverings on τ disjoint sets N_i for $1 \leq i \leq \tau$ and the set $N_0 = \{1,..,\upsilon\}$ which is disjoint to all N_i. Let $\mathcal{B}_i$ be the collection of blocks of covering (σ_i,k,t) on the set N_i. Given any arbitrary p subset A of the set $N = \cup_{i=0}^{\tau}N_i$, we will show that there is a block B in one of the covering designs such that $|B \cap A| \geq t$.
Let $p = \sum_{i=0}^{\tau} p_i$, where $A_i = A \cap N_i$ and $p_i = |N_i \cap A_i|$. Now suppose there is no block B in the union of covering designs such that $|B \cap A| \geq t$; in other words $p_i \leq t-1$ for all $1 \leq i \leq \tau$. Then we would have

$$|A| \leq \tau(t-1) + \upsilon = p - 1.$$

This is a contradiction because $|A| = p$. Therefore there exists an i such that $p_i \geq t$, and consequently there is a block $B \subseteq \mathcal{B}_i$ such that $|B \cap A| \geq t$. This holds for any arbitrary p set $A \subseteq N$. Therefore we have an (n,k,p,t) lotto design.

This applies to any arbitrary set of coverings that satisfy the theorems conditions. This includes the smallest possible coverings that satisfy these conditions. Therefore

$$L(n,k,p,t) \leq \min_{n-\upsilon=\sum_{i=1}^{\tau}\sigma_i} \sum_{i=1}^{\tau} C(\sigma_i,k,t).$$

□

Example 2.12. If we want to find an upper bound for $L(40,6,14,4)$, then we can use theorem 2.11. To do so, we would do the following:

$$14 = \tau(3) + 1 + \upsilon \Rightarrow \tau = 4, \upsilon = 1.$$

This gives us

$$L(40,6,14,4) \leq \sum_{i=1}^{4} C(\sigma_i,6,4),$$

where $\sum_{i=1}^{4}\sigma_i = 39$. Looking up the tables in [4] we see that in this case the sum is minimal when

$$\sigma_1 = \sigma_2 = \sigma_3 = 10, \text{ and } \sigma_4 = 9$$

(or some similar permutation of this). The upper bounds for $C(10,6,4)$ and $C(9,6,4)$ are 20 and 12 respectively [4]. Therefore $L(40,6,14,4) \leq 3 \cdot 20 + 12 = 72$.

Theorem 2.13. *Let $n = n_1 + n_2$, $0 \leq s \leq t$ and $s \leq l \leq k - t + s$. Then*

$$C(n,k,t) \leq \sum_{s=0}^{t} \min_{l} C(n_1,l,s) \cdot C(n_2,k-l,t-s).$$

Proof. Let $N_1 = \{1,..,n_1\}$ and $N_2 = \{n_1+1,..,n_2\}$. Let $\mathcal{B}_{l,s}$ be the set of blocks you get when for each block of a (n_1,l,s) covering design and each block of a $(n_2,k-l,t-s)$ covering design we take the union of these two blocks. Do this for all possible values of l and s. Choose an arbitrary $p = t$ set $A \subseteq N = \{1,..,n\}$ and let $A_1 = A \cap N_1$ and $A_2 = A \cap N_2$. Since $|A_1 \cup A_2| = t$ we know that there is a set of blocks $\mathcal{B}_{l,A_1}$ for each possible value of l. The set $\mathcal{B}_{l,A_1}$ is made from joining all blocks from the (n_1,l,A_1) covering design with all blocks from the $(n_2,k-l,t-A_1)$ covering design. Because of this, there is a block $B \subseteq \mathcal{B}_{l,A_1}$ such that $B = B_1 \cup B_2$, $|A_1 \cap B_1| = s$ and $|A_2 \cap B_2| = t-s$. When we form the union of B_1, and B_2 we see that

$$|A \cap (B_1 \cup B_2)| = |(A_1 \cap B_1) \cup (A_2 \cap B_2)| = (s + (t-s)) = t.$$

Therefore, for any arbitrary $p = t$ set A we could choose there is a block B such that $|B \cap A| = t$. So we have a (n,k,t) covering design. So given

any collection of coverings of the form (n_1, l, s), and $(n_2, k-l, t-s)$ for all possible values of l, and s we can form a (n, k, t) covering design.
This includes the coverings that give us the minimum value for the sum of the products of their size. The minimal number of blocks in this covering design is given by:

$$\begin{aligned} & C(v_1, l, 0) \cdot C(v_2, k-l, t) \\ + \; & C(v_1, l, 1) \cdot C(v_2, k-l, t-1) \\ & \vdots \\ + \; & C(v_1, l, t-1) \cdot C(v_2, k-l, 1) \\ + \; & C(v_1, l, t) \cdot C(v_2, k-l, 0) \end{aligned}$$

Therefore

$$C(n, k, t) \leq \sum_{s=0}^{t} \min_{l} C(v_1, l, s) \cdot C(v_2, k-l, t-s).$$

□

We can try different variations of n_1 and n_2 to get a better upper bound. One thing to note is that if $a + b = n$, then trying $n_1 = a$ and $n_2 = b$ will give us the same result as $n_1 = b$ and $n_2 = a$.

2.2 Algorithms for Constructing Lotto Designs

There are three main types of algorithms used to construct lotto designs. They are greedy algorithms, backtracking, and simulated annealing. Due to space constraints, we will only briefly discus each type of algorithm. During the writing of this paper a greedy algorithm was written and implemented.

2.2.1 Greedy Algorithms

Greedy algorithms are the simplest algorithms for finding lotto designs. They are very easy to program. However, they usually don't find optimal solutions. The way a greedy algorithm works is by finding a feasible solution by choosing the best option at each step. So for a lotto design, at each step the algorithm would choose the ticket that covers the most p sets that are so far not covered by any other ticket. Presented below is some basic pseudo code for a greedy lotto design algorithm.

Greedy($\mathcal{P}$)
Input: $\mathcal{P} = [p_0, p_1, ..., p_m]$,profit function
Output: $X = [x_0, x_1, ..., x_l]$, a subset of $\mathcal{P}$
Profit = 0
$X = \emptyset$
foreach $p_i \in \mathcal{P}$ **do**
  if *profit*(p_i) >*Profit* **then**
    Point = p_i
    Profit = profit(p_i)
  end
end
$\mathcal{P} \leftarrow \mathcal{P}\backslash$Point.
$X \leftarrow X \cup$ Point
Greedy($\mathcal{P}$)

Algorithm 1: Pseudo Code for a Greedy Algorithm

In this code the set $\mathcal{P}$ is the set of all the possible k sets for the lotto design and the profit function is the number of uncovered p sets each k set covers. As the code runs, it finds the k set (called Point) that covers the most uncovered p sets, it then adds it to the set X, then runs again with the set $\mathcal{P}\backslash$Point and a new profit function. It will keep doing this until all the p sets have been covered. In the appendix is a greedy algorithm written in Maple.

2.2.2 Backtracking Algorithms

Backtracking is an exhaustive search method, it considers every feasible solution. Because of this, it will always find an optimal solution. The draw back of this is that it consumes large amounts of memory, and can take a long time to run. Certain methods called *pruning methods* can be used to decrease the workload. Presented below is some pseudo code based on the code from [5], pg 107 for a general backtracking algorithm. $\mathcal{P}$ is called the *possibility set*, it is made up of all the possible solutions the problem could have. In the case of a lotto design it is made up of all possible tickets. For each partial solution $X = [x_0, x_1, ..., x_{l-1}]$ there is a *choice set* $C_l \subseteq \mathcal{P}_i$ that contains all feasible points from $\mathcal{P}_i$ given that $X = [x_0, x_1, .., x_{l-1}]$. Using the choice set is one pruning method because at each step, you don't have to consider all of the x_i's remaining. You just consider the $x_i \in C_l$.

BACKTRACK(l)
Input: x, constraints
Output: X
if $[x_0, x_1, ...]$ *is a feasible solution* **then**
| *process it*
end
compute C_l
foreach $x \in C_l$ **do**
| $x_l \leftarrow x$,
| $BACKTRACK(l+1)$
end

Algorithm 2: Pseudo Code for a Backtracking Algorithm

To start off, $l = 0$, and $X = \emptyset$. The basic idea of the algorithm is if X is a feasible solution then the process command checks if it is better then the current best solution, and saves it if it is. If it is not a feasible solution it creates new solutions by adding each x from the choice set to the current partial solution X and runs the algorithm again with the new X. Once it has gone through all possible solutions it outputs the current best solution X.

2.2.3 Simulated Annealing

Simulated Annealing is a modified version of *Hill-climbing*. The way hill-climbing algorithms work is they start with a feasible solution and try to improve it to get an optimal solution. The way this is done is by taking the current solution and looking at the solutions in its *neighborhood*, checking to see if any of these solutions are better. The neighborhood of a solution is all the different solutions possible by slightly changing the current solution. The problem with hill-climbing is that is can get stuck at a local maximum. This will happen if the current solution is better then all other solutions in its neighborhood.

Simulated annealing is a way of getting round this problem. In simulated annealing even if the solution being checked is worse then the current one there is a probability that this solution will be still accepted. This is how simulated annealing avoids getting stuck at local maximums. The probability that a solution is chosen despite not being as good as the current solution is often referred to as the temperature, and denoted by T. As the algorithm runs, the temperature T decreases. Usually simulated annealing will run until it reaches its iteration limit, at which point the current solution is outputted. Below is some pseudo code for simulated annealing. In the code $N(X)$ performs a neighborhood search and chooses a random neighbor of X. It returns fail if, then there are no points in the neighborhood of X.

Annealing(c_{max}, T_0, α, X)
Input: c_{max}, T_0, α, X
Output: X_{best}
$c = 0$
$T = T_0$
$X_{best} = X$
while $c \leq c_{max}$ **do**
  $Y = N(X)$
  if $Y \neq Fail$ **then**
    if $P(Y) > P(X)$ **then**
      $X = Y$
      if $P(X) > P(X_{best})$ **then**
        $X_{best} = X$
      end
    else
      $r = Random(0, 1)$
      if $r < e^{(P(Y)-P(X))/T}$ **then**
        $X = Y$
      end
    end
  end
  $c = c + 1$
  $T = \alpha T$
end
return X_{best}

Algorithm 3: Pseudo Code for a Simulated Annealing Algorithm

In the algorithm c records the number of iterations and c_{max} is the maximum number of iterations. the algorithm starts off with a feasible solution X it checks a random feasible solution from the current solution's *neighborhood*. If this solution is better then the current one then it becomes the current solution. If not, then it chooses a random number between 0 and 1. If this number is less then $e^{(P(Y)-P(X))/T}$ then it becomes the current solution anyway. At the end of each iteration the temperature T is decreased by a factor of α. Once it reaches its iteration limit it outputs the current best solution.

CHAPTER 3

Lotto Designs on the NZ Lotto System

3.1 The NZ Lottery

In the NZ lotto system there are 40 balls and 6 are chosen at random. After this a final bonus ball is chosen at random from the remaining 34 numbers. Below is a table of the possible winning tickets and how much you can expect to win[1]. For more information see [1]

	Number of Winning Numbers	Bonus Ball	Average Winnings	Corresponding Lotto Design
Division 1	6	No	\$833,333	(40,6,6,6)
Division 2	5	Yes	\$27,587	(40,6,7,6)
Division 3	5	No	\$699	(40,6,6,5)
Division 4	4	Yes	\$64	(40,6,7,5)
Division 5	4	No	\$34	(40,6,6,4)
Division 6	3	Yes	\$24	(40,6,7,4)

We want to be able to construct lotto designs corresponding to these six divisions. The obvious goal is be able to construct a lotto design for one (or more) of these divisions such that the amount of money required to buy all the tickets in the design is less then the amount of money we can expect to win.

The first step is to calculate upper and lower bounds for these divisions. Calculating upper and lower bounds for a design is much easier to do then to calculate the actual lotto design. If we discover that the lower bound for a lotto design is greater the maximum number of tickets we can by and still make a profit, then there is no point in trying to construct that lotto design. Using proposition 2.9 and theorem 2.10 we can construct lower bounds for the NZ lotto system. With the power ball we have:

$$\begin{aligned} 62145 &\leq \frac{T(40,7,6)}{\binom{6}{6}} \leq L(40,6,7,6) \\ 6906 &\leq \frac{T(40,7,5)}{\binom{6}{5}} \leq L(40,6,7,5) \\ 280 &\leq \frac{T(40,7,4)}{\binom{6}{4}} \leq L(40,6,7,4) \end{aligned}$$

[1] Based on the lotto draws over the last 10 weeks

And without making use of the power ball we have:

$$\begin{aligned} 3838380 \leq & \ \frac{T(40,6,6)}{\binom{6}{6}} \leq L(40,6,6,6) \\ 21325 \leq & \ \frac{T(40,6,5)}{\binom{6}{5}} \leq L(40,6,6,5) \\ 433 \leq & \ \frac{T(40,6,4)}{\binom{6}{4}} \leq L(40,6,6,4) \end{aligned}$$

Unfortunately, now that we have calculated lower bounds for all six divisions we see that no possible lotto design could guarantee us a profit, as the table below shows. This table is just for the lower bounds, the actual designs could cost us a lot more.

	Cost	Guaranteed Average Winnings	Profit
Division 1	\$2,303,028	\$833,333	−\$1,469,695
Division 2	\$37,287	\$27,587	−\$9,700
Division 3	\$12,795	\$699	−\$12,096
Division 4	\$4,143.6	\$64	−\$4,079.6
Division 5	\$259.8	\$34	−\$225.8
Division 6	\$168	\$24	−\$144

Even though none of these lotto designs can be of use to us, we would still like to calculate some upper bounds for them. These are generally a lot trickier to calculate. We will use a number of different theorems to try to get the lowest upper bound.

Not all of the cases we are interested in are difficult. The upper bound for $L(40,6,6,6)$, (which corresponds to winning first division), is very easy to calculate. To have a $(40,6,6,6)$ lotto design, we require that choosing sets of 6, every possible set of 6 is contained in one of the sets we chose. The only way to achieve this is to choose every possible set of 6. So $L(40,6,6,6) = \binom{40}{6} = 3838380$. So the upper bound is (trivially) 3838380. That is,

$$L(40,6,6,6) \leq 3838380.$$

Another easy upper bound is $L(40,6,7,4)$, which would guarantee a sixth division win. Using theorem 2.7, varying the values of n_1, and n_2 we get $L(40,6,7,4) \leq L(20,6,4,4)+L(20,6,4,4)$. One thing to note is that $L(40,6,4,4)$ is a covering design. This is good because we have access to some good tables on covering designs in [4]. From the tables in [4] we know that $L(20,6,4,4) \leq 456$. This gives us our second upper bound,

$$L(40,6,7,4) \leq 912.$$

Now, using theorem 2.11, we know that $L(40,6,7,6) \leq C(39,6,6)$. Using a similar argument as we did for $L(40,6,6,6)$, we deduce that $L(40,6,7,6) \leq C(39,6,6) = \binom{39}{6} = 3262623$. So now we have our third upper bound.

$$L(40,6,7,6) \leq 3262623$$

The final three upper bounds require a bit more work. Using theorems 2.11 and 2.13 we can construct upper bounds for the remaining lotto designs, $L(40,6,7,5)$, $L(40,6,6,5)$, and $L(40,6,6,4)$. These are

$$\begin{aligned} L(40,6,7,5) &\leq 105339 \\ L(40,6,6,5) &\leq 123478 \\ L(40,6,6,4) &\leq 7474 \end{aligned}$$

There is quits a lot of tedious work involved in these calculations. See the appendix for an example of the calculations involved in calculating $L(40,6,6,4)$. Now we have complete lower and upper bounds for all six divisions of the NZ lotto system. With the bonus ball we have:

$$\begin{array}{lll} \text{Division 2}: & 62145 \leq L(40,6,7,6) & \leq 3262623 \\ \text{Division 4}: & 6906 \leq L(40,6,7,5) & \leq 105339 \\ \text{Division 6}: & 280 \leq L(40,6,7,4) & \leq 912 \end{array}$$

And without the bonus ball.

$$\begin{array}{lll} \text{Division 1}: & 3838380 \leq L(40,6,6,6) & \leq 3838380 \\ \text{Division 3}: & 21325 \leq L(40,6,6,5) & \leq 123478 \\ \text{Division 5}: & 433 \leq L(40,6,6,4) & \leq 7474 \end{array}$$

3.2 Lucky Numbers

Some people foolishly or otherwise think that some numbers are lucky, ie have more chance of being drawn then regular numbers. If this is the case then you can drastically reduce the number of tickets needed to get a t match. Of course in these situations you lose the guarantee of getting a t match. But if you believe that the lucky numbers are suitably likely to be drawn then you can construct lotto designs with a positive expected gain.

Example 3.1. Working with the NZ lotto system, assume that you believe that 16 out of the 40 numbers are lucky. We will assume that there is a high enough chance that of the 7 numbers drawn, all of them will come from these 16 lucky numbers. If these numbers were not lucky, then this would only happen about 0.06% of the time. To get a 4 match (and hence win sixth division) we will need an $L(16,6,7,4)$ lotto design. Using theorem 2.10 (and theorem 2.9) we can construct a lower bound for this design.

$$\begin{aligned} L(16,6,7,4) &\geq \frac{T(16,7,4)}{\binom{6}{4}} \\ &= \frac{\binom{16}{4}}{\binom{6}{3}\cdot\binom{6}{4}} \cdot \frac{16-7+1}{16-4+1} \\ &= \frac{1820}{300} \cdot \frac{10}{13} \\ &= 5. \\ \text{Therefore } L(16,6,7,4) &\geq 5. \end{aligned}$$

This gives a lower bound of 5. So with each ticket costing \$0.6, this design would cost a minimal of $\$0.6 \cdot 5 = \$3$. With the average return for winning sixth division being around \$24, you could make a profit if all of the numbers drawn were from your lucky numbers occurred about every eighth draw. However, this would mean that it was happening about 200 times more often then it should be. So for this to be the case your lucky numbers would have to be very lucky. This is just a lower bound, we may need to buy more tickets then this to get our design.
From [6] we get an upper bound of 14 for a $(16, 6, 7, 4)$ lotto design. This would cost you \$8.4. If $L(16, 6, 7, 4)$ was 14 then you would need all of the numbers drawn to be from your set of lucky numbers happening every third time. This is about 540 times higher then what you would expect normally. Your numbers would have to be extremely lucky for this to happen.
If we want to construct an $(16, 6, 7, 4)$ lotto design there are several ways we can do this. One way is to first use theorem 2.7. This tells us that $L(16, 6, 7, 4) \leq L(8, 6, 4, 4) + L(8, 6, 4, 4)$. Doing this make it a lot easier because it is a lot easier to calculate an $(8, 6, 4, 4)$ lotto design then it is to calculate a $(16, 6, 7, 4)$ lotto design. The following $(8, 6, 4, 4)$ lotto design was constructed by the greedy algorithm in the appendix. The blocks are:

$\{1, 2, 3, 4, 5, 6\}, \{1, 2, 3, 4, 7, 8\}, \{1, 2, 5, 6, 7, 8\}, \{3, 4, 5, 6, 7, 8\}, \{1, 2, 3, 4, 5, 7\},$
$\{1, 2, 3, 4, 5, 8\}, \{1, 2, 3, 4, 6, 7\}, \{1, 2, 3, 4, 6, 8\}.$
We can join this lotto design together with a similar covering on $\{9, 10, 11, 12, 13, 14, 15, 16\}$. This will give a $(16, 6, 7, 4)$ lotto design with the following blocks:

$\{1, 2, 3, 4, 5, 6\}, \{1, 2, 3, 4, 7, 8\}, \{1, 2, 5, 6, 7, 8\}, \{3, 4, 5, 6, 7, 8\}, \{1, 2, 3, 4, 5, 7\},$
$\{1, 2, 3, 4, 5, 8\}, \{1, 2, 3, 4, 6, 7\}, \{1, 2, 3, 4, 6, 8\}\{9, 10, 11, 12, 13, 14\},$
$\{9, 10, 11, 12, 15, 16\}, \{9, 10, 13, 14, 15, 16\}, \{11, 12, 13, 14, 15, 16\}, \{9, 10, 11, 12, 13, 15\},$
$\{9, 10, 11, 12, 13, 16\}, \{9, 10, 11, 12, 14, 15\}, \{9, 10, 11, 12, 14, 16\}.$

Example 3.2. Once again we will be working with the NZ lotto system, and the same set of lucky numbers as example 3.1. This time we will look at $L(16, 6, 7, 5)$, which corresponds to winning fourth division(4 numbers from the 6 drawn plus the bonus ball). The odds of all 7 balls drawn being from our set of lucky numbers is around 0.06%. The average winnings for fourth division is about \$64. Again using theorem 2.10 we can construct a lower bound for $L(16, 6, 7, 5)$, which is 41. To use this design it would cost us a minimum \$24.6, so to make a profit the odds of all 7 balls being drawn from our set of 16 lucky numbers must be greater then around 61%. This is roughly 1000 times higher then expected odds. This is also the best case scenario, from theorem 2.11 and [4] we find that the upper bound is 385. It would cost \$231 to implement this design, about 3.6 times more then what you could win. In this case, the design would be very impractical.

CHAPTER 4

Conclusion

Winning lotto is what dreams are made of. But we have seen that winning is not an easy task. While it can be easy to construct small lotto designs, constructing large lotto designs ($n > 30$ say) is very difficult. This is what lotteries rely on. Otherwise everybody would be constructing lotto designs. Despite the difficulty, this is still an active area of research. In this paper we saw only a small part of the number of theorems and constructions related to lotto designs.

In chapter 3 we constructed lower and upper bounds for the NZ lotto system. While these were not optimal bounds, they still gave us an idea of the number of tickets needed for the corresponding lotto designs. In particular, we saw that the possible winnings for the NZ lotto system were well below the value needed to make playing lotto practical. That is, the number of tickets needed to make an optimal lotto design for any one of the divisions was far too high when compared with the amount we could win. However, this is just one lottery, there may be other lotteries out there for which effective lotto designs can be constructed.

APPENDIX A

Calculating the Upper Bound for L(40,6,6,4)

To calculate the upper bound for $L(40,6,6,4)$, the first thing we need to do is use proposition 2.11 to write $L(40,6,6,4)$ as the sum of suitable coverings. When we do this we get the following results.

$$\begin{aligned} p &= \tau(t-1)+1+\upsilon \\ 6 &= \tau(4-1)+1+\upsilon \\ 5 &= 3\tau+\upsilon \end{aligned}$$

Therefore $\tau = 1, \upsilon = 2$. So we have $L(40,6,6,4) \leq C(38,6,4)$. Now we use proposition 2.13 to calculate an upper bound. For the best results we have to check all values of n_1 such that $19 \leq n_1 \leq 32$. We don't check for higher values of n_1 because then n_2 would be less then k, which is not possible for a covering. After we do a few calculations we will come to $n_1 = 27$, and $n_2 = 11$. This gives us the smallest upper bound. The following calculations were used to calculate the upper bound.

min of	$C(27,0,0)\cdot C(11,6,4)$	$=1\cdot 32$	$=32$	
	$C(27,1,0)\cdot C(11,5,4)$	$=1\cdot 66$	$=66$	
	$C(27,2,0)\cdot C(11,4,4)$	$=1\cdot 330$	$=330$	min = 32
+ min of	$C(27,1,1)\cdot C(11,5,3)$	$=27\cdot 20$	$=540$	
	$C(27,2,1)\cdot C(11,4,3)$	$=14\cdot 47$	$=658$	
	$C(27,3,1)\cdot C(11,3,3)$	$=9\cdot 165$	$=1485$	min = 540
+ min of	$C(27,2,2)\cdot C(11,4,2)$	$=351\cdot 11$	$=3861$	
	$C(27,3,2)\cdot C(11,3,2)$	$=117\cdot 19$	$=2223$	
	$C(27,4,2)\cdot C(11,2,2)$	$=61\cdot 55$	$=3355$	min = 2223
+ min of	$C(27,3,3)\cdot C(11,3,1)$	$=2925\cdot 4$	$=11700$	
	$C(27,4,3)\cdot C(11,2,1)$	$=763\cdot 6$	$=4578$	
	$C(27,5,3)\cdot C(11,1,1)$	$=319\cdot 11$	$=3509$	min = 3509
+ min of	$C(27,4,4)\cdot C(11,2,0)$	$=17550\cdot 1$	$=17550$	
	$C(27,5,4)\cdot C(11,1,0)$	$=3906\cdot 1$	$=3906$	
	$C(27,6,4)\cdot C(11,0,0)$	$=1170\cdot 1$	$=1170$	min = 1170
				total=7474

From this we have $L(40,6,6,4) \leq 7474$. This is the smallest upper bound possible using this approach.

APPENDIX B

Greedy Algorithm in Maple

What follows is a greedy algorithm to construct lotto designs written in maple. The code is usually in one piece, but has been broken up to allow for comments explaining what it is doing.

```
greedy:=proc(n,k,p,t)
local lp,mp,lp2,lp3,lk,mk,cover,total,P,count,j,
com,ii,jj,hh,gg,iteration,winner,h,rank,z;
with(combinat, choose);
with(combinat,numbcomb);
```

lp is the set of all possible p sets.

```
lp:=choose(n,p);
mp:=numbcomb(n,p);
lp2:=array(1..mp,1..1);
lp3:=array(1..mp,1..1);
```

lk is the set of all possible k sets.

```
lk:=choose(n,k);
mk:=numbcomb(n,k);
```

cover is the maximum number of p sets any k set can cover.

```
cover:=0;
```

total is the total number of blocks in the design.

```
total:=1;
P:=numbcomb(p,t);
```

count is the total number of p sets that are covered at each point of the code.

```
count:=0;
```

rank keeps track of how many uncovered p sets each k set covers at each iteration of the code.

```
rank:=Matrix(mk,1);
```

The next part of the code checks to see the maximum number of p sets each k set can cover.

```
for j from 1 to mp do; com:=choose(lp[j],t);
  for ii from 1 to P do;
    if ((convert(com[ii],set) subset convert(lk[1],set))=true) then;
    cover:=cover+1;
    break;
    end if;
  end do;
end do;
```

Before any k set is chosen, every possible k set covers the maximum number (cover) of uncovered p sets.

```
for jj from 1 to mk do;
rank[jj,1]:=cover;
end do;
```

The first block the code chooses is $\{1, ..., k\}$.

```
print(convert(lk[1],set));
count:=cover;
```

Now the code adjusts the number of uncovered p sets that each possible k set covers according to the p sets covered by the first block, and also sets lp2[j,1] to 0 if the p set j is now covered.

```
for jj from 2 to mk do;
  for j from 1 to mp do;
  com:=choose(lp[j],t);
    for hh from 1 to P do;
      if ((convert(com[hh],set) subset convert(lk[1],set))=true) then;
        for gg from 1 to P do;
          if (convert(com[gg],set) subset convert(lk[jj],set)=true) then;
          rank[jj,1]:=rank[jj,1]-1;
          break;
          end if;
        end do;
        lp2[j,1]:=0;
        break;
      end if;
    end do;
  end do;
end do;
```

Now the code runs through 100 iterations choosing the best k set at each iteration until all p sets are covered. For larger designs we would need to change the upper limit of iterations to something greater then 100.

```
for iteration from 1 to 100 do;
  if (count<mp) then;
```

The first step is to find the unchosen k set that covers the most uncovered p sets.

```
winner:=2;
  for h from 2 to mk do:
    if (rank[h,1]>=rank[winner,1]) then;
    winner:=h;
    end if;
  end do;
  print(convert(lk[winner],set));
```

Once the best k set is chosen, count increases by the number of uncovered p sets the k set covers, then the number of uncovered p sets this k set covers is set to 0.

```
count:=count+rank[winner,1];
total:=total+1;
rank[winner,1]:=0;
```

Now the number of uncovered p sets that each k set covers is adjusted according to the p sets that the k set covered.

```
    for jj from 2 to mk do;
      for j from 1 to mp do;
```

First we check to see if the current k set covers any uncovered p sets.

```
        if (rank[jj,1]<>0) then;
```

If it does, then we check to see if each uncovered p set is covered by the k set.

```
          if (lp2[j,1]<>0) then;
          com:=choose(lp[j],t);
            for z from 1 to P do;
              if ((convert(com[z],set) subset convert(lk[winner],set))=true)
              then;
              lp3[j,1]:=1;
                for ii from 1 to P do;
                  if ((convert(com[ii],set) subset convert(lk[jj],set))=true)
                  then;
                  rank[jj,1]:=rank[jj,1]-1;
                  break;
                  end if;
                end do;
              break;
              end if;
            end do;
```

```
            end if;
          end if;
        end do;
      end do;
      for j from 1 to mp do;
        if (lp3[j,1]=1) then;

        lp2[j,1]:=0;
        end if;
    end do;
  end if;
end do;
print(total);
end proc:
```

It may seem strange to force the algorithm to choose the first block to be $\{1,..,k\}$, but this has no effect on the efficiency of the algorithm. Through a suitable isomorphism we can change the block $\{1,..,k\}$ to any other k subset of $N = \{1,..,n\}$. The fact that the first block has points labeled 1 to k does not change the size or effectiveness of the design.

Bibliography

[1] *NZ Lotteries*, http://www.nzlotteries.co.nz/wps/wcm/myconnect/lotteries2/nzlotteries/, Date accessed:05-02-08.

[2] D Caen, *Extension of a Theorem of Moon and Moser on Complete Subgraphs*, Ars Combin **16** (1983), 5–10.

[3] Charles Colbourn and Jeffery Dinitz, *The CRC Handbook of Combinatorial Designs*, CRC Press, 1996,.

[4] Daniel M. Gordon, Oren Patashnik, and Greg Kuperberg, *New Constructions for Covering Designs*, Journal of Combinatorial designs (1995), 269–284.

[5] Donald L. Kreher and Douglas R. Stinson, *Combinatorial Algorithms*, The CRC Press Series on Discrete Mathematics and its Applications, CRC Press, 1999.

[6] Pak Ching Li and G. H. John Van Rees, *Lotto Design Tables*, citeseer.ist.psu.edu/680238.html, Date accessed 06-02-08.

[7] ———, *Lower Bounds on Lotto Designs*, Congressus Numerantium **141** (1999), 5–30.

[8] ———, *New Constructions of Lotto Designs*, Utilitas Mathematica **58** (2000), 45–64.

[9] Kari Nurmela and Patric Ostergard, *Upper Bounds for Covering Designs by Simulated Annealing*, Congressus Numerantium **96** (1993), 93–111.